Stuart School of Business
ILLINOIS INSTITUTE OF TECHNOLOGY

Teach for America Case Study

MAX 522: Predictive Analytics
Professor Dinakar Jayarajan

Group 3:
Ana Manuela Bustamante Vela
Carolina Guerrero Requejo
Nihaal Karkera
Pauline Punio
Ana Uribarri
Tachin Ho

Table of Contents

I. Project Overview

In 2016, Teach For America (TFA), a nonprofit organization dedicated to reducing educational inequity, faced a significant challenge. TFA's annual corps size had grown to over 4,000, yet applications had declined from a peak of 57,000 in 2013. This downward trend raised concerns about TFA's ability to sustain its impact and achieve its mission of bringing diverse, high-achieving leaders to serve as teachers in low-income communities across the U.S.

To address these challenges, TFA's Recruitment and Admissions teams, led by Senior Vice President Josh Anderson, reevaluated their strategies. Michael Metzger, a senior manager with a background in computer science, spearheaded the development of predictive models aimed at enhancing TFA's recruitment and admissions process. Metzger's work focused on identifying promising applicants, assessing withdrawal risks, and refining recruitment techniques using data-driven approaches.

In pursuit of these goals, Metzger and his team compiled and cleaned a dataset from the 2015-2016 recruitment cycle. This dataset included variables such as essay word counts, sentiment scores, application milestones, and event attendance. These factors informed predictive models designed to anticipate applicant behavior, with the aim of helping recruiters focus on high-potential applicants.

This project applies machine learning classifiers—Naive Bayes, Decision Tree, Artificial Neural Networks (ANN), and Support Vector Machine (SVM) to classify applicants based on their likelihood of completing the admissions process. Through this analysis, we aim to develop a data-backed strategy to improve its recruitment yield, reduce applicant attrition, and ultimately support its mission to address educational inequity.

The objective of this project is to develop a predictive model to assess the risk that future applicants might withdraw from consideration during the selection process. By leveraging machine learning classifiers such as Naive Bayes, Decision Tree, Artificial Neural Networks (ANN), and Support Vector Machine (SVM), we aim to analyze applicant data from the 2015-2016 recruitment cycle to identify patterns and predictors of withdrawal. This model will enable Teach For America (TFA) to proactively address potential risks, enhance its recruitment strategies, and allocate resources more effectively toward high-potential candidates. Ultimately, this approach supports TFA's mission to reduce educational inequity by ensuring a steady pipeline of committed and diverse leaders to serve in low-income communities.

II. Data clean-up procedure

In this project, we developed four different predictive models: Naive Bayes, Decision Tree, SVM, and Artificial Neural Network (ANN). Each of these models has specific requirements regarding

the type of data they use. Consequently, the data cleaning procedure was divided into two main stages: an initial clean-up that was common to all models and then a specific preparation tailored to each set of models based on their characteristics.

Initial Clean-Up (Common to All Models)

The first phase of the data cleaning process was common to all four models and included the following steps:

1. *Removal of Non-Relevant Columns:*

We removed the columns personid, appyear, and schoolsel_chr. These variables did not provide significant predictive value to the dataset. Additionally, schoolsel_chr was already represented in numerical format by the schoolsel variable.

2. *Handling Missing Values (NA):*

Missing values were identified in the schoolsel variable. We decided to replace all missing (NA) values with 0, allowing us to clearly identify these entries without introducing complex biases. This strategy was implemented to ensure uniformity across the dataset for all models.

Model-Specific Data Preparation

After the initial clean-up, specific preparations were conducted for each group of models. Since Naive Bayes and Decision Tree models rely on categorical data, while SVM and ANN models require numerical data, we split the preparations into these two groups.

1. Naive Bayes and Decision Tree

The data preparation for Naive Bayes and Decision Tree models was conducted almost simultaneously since both models require similar types of data, specifically categorical variables.

First, all relevant variables were transformed into categorical factors. This included converting text-based and numeric categorical variables into factors, which allows the models to interpret these categories appropriately during training. The following variables were factorized: stem, schoolsel, major1, major2, major1group, major2group, minor, minorgroup, undergrad_uni, appdeadline, attendedevent, completedadm.

Then, several numeric variables were converted into categorical ranges:

- Essay Lengths: The variables essay1length, essay2length, and essay3length were categorized into four levels based on length ranges (e.g., 1-75 words, 76-150 words, etc.) and then factorized (essay1, essay2, essay3).
- Unique Words in Essays: The number of unique words in the essays (essayuniquewords) was categorized into five levels, then factorized (essayUW).

- Essay Sentiment: The sentiment of the essays (essayssentiment) was categorized into four levels based on the sentiment value range and factorized (essentiment).
- Application Timing: Variables related to the timing of the application (signupdate, starteddate, submitteddate) were categorized into levels and then factorized (signup, started, submitted).
- GPA: The GPA variable (gpa) was categorized into four levels (fgpa), representing different ranges of GPA values, and then factorized.

After transforming these continuous variables into categorical ones, we removed the original numeric columns (gpa, essay1length, essay2length, essay3length, essayuniquewords, essayssentiment, signupdate, starteddate, submitteddate) to ensure that only categorical variables remained for these models.

2. SVM and Artificial Neural Network (ANN)

Beyond the initial cleanup, the SVM model removed irrelevant features such as essay lengths and date-related columns. Key categorical variables like appdeadline and completedadm were converted to factors, with appdeadline ordinally encoded. The submitteddate variable was converted to numeric, and numeric features like gpa, essayuniquewords, essayssentiment, and submitteddate were standardized for consistent scaling. Further information on the data clean up can be found in the Appendix for the SVM R Code.

3. Artificial Neural Network (ANN)

For the ANN model, the data preparation focused on ensuring all features were numerical, as required by neural network algorithms. After the initial cleanup, categorical variables such as appdeadline and completedadm were encoded using one-hot encoding to create binary indicator variables for each category. This transformation ensured that the categorical variables were represented in a way suitable for the ANN's processing.

Numeric variables, including gpa, essayuniquewords, essayssentiment, and submitteddate, were standardized to have a mean of zero and a standard deviation of one. This standardization ensured that all features were on a consistent scale, preventing dominance of features with larger ranges during the training process.

Date-related columns such as signupdate, starteddate, and submitteddate were transformed into numeric formats representing the number of days since a fixed reference date. These numeric representations were then scaled as part of the standardization process.

Finally, unnecessary columns, such as raw text-based essay variables and intermediate categorical variables (e.g., factorized versions used for other models), were removed, leaving a clean, fully numerical dataset ready for input into the ANN model. Further technical details on the data transformation process are provided in the Appendix for the ANN R Code.

III. Naive Bayes

For the Naive Bayes classification, multiple models were evaluated with different subsets of features to determine the configuration that yielded the best predictive performance. Each model's effectiveness was assessed through various metrics, including accuracy, sensitivity, and specificity, derived from the resulting confusion matrices.

First Model: All Predictors

Initially, a Naive Bayes model was constructed using all available predictors. The results are shown in Figure 3-1.

Naive Bayes: Confusion Matrix and Performance Statistics

Reference / prediction	0	1
0	81	142
1	1158	4868

- **Accuracy**: 0.792 (95% CI: 0.7817, 0.802)
- **No Information Rate**: 0.8017
- **Kappa**: 0.0536
- **McNemar's Test P-Value**: < 2e-16
- **Sensitivity**: 0.06538 (ability to identify applicants likely to withdraw)
- **Specificity**: 0.97166 (ability to identify applicants likely to complete)
- **Positive Predictive Value**: 0.36323
- **Negative Predictive Value**: 0.80783
- **Balanced Accuracy**: 0.51852

Figure 3-1: First Model Naive Bayes Confusion Matrix and Performance Statistics

The model showed a reasonably high specificity, meaning it accurately identifies applicants who are likely to complete the process. However, the sensitivity was relatively low, indicating that the model did not perform well at identifying applicants likely to withdraw.

The high specificity suggests that the model is effective at correctly identifying applicants who will complete the process, while the lower sensitivity indicates a limitation in recognizing those who might withdraw. This may limit the model's ability to support initiatives aimed at preventing withdrawals.

Second Model

In the second Naive Bayes model, a subset of predictors was chosen to streamline the analysis and potentially improve interpretability. The variables used in this model were: *stem*, *schoolsel*,

major1group major2group, *minorgroup*, *appdeadline*, *attendedevent*, *essay1*, *essay2*, *essay3*, *essayUW*, *essentiment*, *signup*, *started*, *submitted*, and *fgpa*. The goal was to include predictors that were presumed to have the most significant impact on applicant behavior, while also reducing unnecessary complexity from the full feature set.

Naive Bayes: Confusion Matrix and Performance Statistics

Reference / prediction	0	1
0	30	38
1	1209	4972

- **Accuracy**: 0.8004 (95% CI: 0.7903, 0.8103)
- **No Information Rate**: 0.8017
- **Kappa**: 0.0258
- **McNemar's Test P-Value**: < 2e-16
- **Sensitivity**: 0.024213
- **Specificity**: 0.992415
- **Positive Predictive Value**: 0.441176
- **Negative Predictive Value**: 0.804401
- **Balanced Accuracy**: 0.508314

Figure 3-2: Second Model Naive Bayes Confusion Matrix and Performance Statistics

The performance of this model, as seen in Figure 3-2, showed slight changes compared to the first model. This model exhibited slightly higher accuracy compared to the first model. However, sensitivity remained low, which means the model still struggles to identify applicants who are likely to withdraw. The specificity is very high, indicating good performance in predicting completions.

Third Model

In the third Naive Bayes model, an even more selective subset of predictors was used in order to further simplify the model and potentially reduce overfitting. This subset excluded *signup* and *started* from the previous iteration, leaving the following variables: *stem*, *schoolsel*, *major1group*, *major2group*, *minorgroup*, *appdeadline*, *attendedevent*, *essay1*, *essay2*, *essay3*, *essayUW*, *essentiment*, *submitted*, and *fgpa*.

Naive Bayes: Confusion Matrix and Performance Statistics

Reference / prediction	0	1
0	22	26
1	1217	4984

- **Accuracy**: 0.8011 (95% CI: 0.791, 0.8109)
- **No Information Rate**: 0.8017

- **Kappa**: 0.0197
- **McNemar's Test P-Value**: < 2e-16
- **Sensitivity**: 0.017756
- **Specificity**: 0.994810
- **Positive Predictive Value**: 0.458333
- **Negative Predictive Value**: 0.803741
- **Balanced Accuracy**: 0.506283

Figure 3-3: Third Model Naive Bayes Confusion Matrix and Performance Statistics

The results of this third model, presented in Figure 3-3, show similar accuracy to the previous one, with slightly lower sensitivity and very high specificity. As with the other models, sensitivity remains a concern, indicating that it is not very effective at identifying those likely to withdraw from the process.

Fourth Model

In the fourth iteration of the Naive Bayes model, *trainControl* was employed to perform cross-validation and improve model robustness. A subset of predictors was selected, similar to previous models, but this time a 10-fold cross-validation approach was used to provide a more stable estimate of the model's performance. The variables included were: *stem*, *schoolsel*, *major1group*, *major2group*, *minorgroup*, *appdeadline*, *attendedevent*, *essay1*, *essay2*, *essay3*, *essayUW*, *essentiment*, *signup*, *started*, *submitted*, and *fgpa*.

Naive Bayes: Confusion Matrix and Performance Statistics

Reference / prediction	0	1
0	80	110
1	1159	4900

- **Accuracy**: 0.7969 (95% CI: 0.7867, 0.8068)
- **No Information Rate**: 0.8017
- **Kappa**: 0.0625
- **McNemar's Test P-Value**: < 2e-16
- **Sensitivity**: 0.06457
- **Specificity**: 0.97804
- **Positive Predictive Value**: 0.42105
- **Negative Predictive Value**: 0.80871
- **Balanced Accuracy**: 0.52131

Figure 3-4: Fourth Model Naive Bayes Confusion Matrix and Performance Statistics

The fourth model, as shown in Figure 3-4, performed slightly worse in terms of accuracy but exhibited a more rigorous approach through cross-validation. Sensitivity remained low, while specificity stayed strong, indicating the model was still primarily effective at predicting completions.

Fifth Model

In the fifth and last model, the focus shifted to optimizing the Naive Bayes classifier through hyperparameter tuning. Using a *trainControl* setup, a search grid was created to explore various values for parameters such as *usekernel*, *fL*, and *adjust*. The parameters were tuned to determine the most effective configuration that could potentially improve model performance.

Naive Bayes: Confusion Matrix and Performance Statistics

Reference / prediction	0	1
0	81	112
1	1158	489

- **Accuracy**: 0.7968 (95% CI: 0.7866, 0.8067)
- **No Information Rate**: 0.8017
- **Kappa**: 0. 0631
- **McNemar's Test P-Value**: < 2e-16
- **Sensitivity**: 0.06538
- **Specificity**: 0. 97764
- **Positive Predictive Value**: 0. 41969
- **Negative Predictive Value**: 0. 80878
- **Balanced Accuracy**: 0. 52151

Figure 3-5: Fifth Model Naive Bayes Confusion Matrix and Performance Statistics

As shown in Figure 3-5, this model did not exhibit a substantial increase in accuracy compared to the previous versions, and sensitivity continued to lag behind, indicating a consistent issue with identifying applicants likely to withdraw. However, specificity remained high, suggesting the model reliably predicted applicants likely to complete the process.

Conclusion and Recommendations

The models analyzed in this case study demonstrated strong specificity, effectively identifying applicants who were likely to complete the process. However, all models faced challenges with sensitivity, indicating difficulty in predicting applicants who would withdraw. The fourth model, which incorporated a more rigorous training process with cross-validation and parameter tuning, showed the best overall performance, making it the most suitable for balancing these metrics. Improving sensitivity further would provide recruiters with better insights into candidates most likely to withdraw, thereby enhancing targeted recruitment efforts.

While the models can be valuable in pinpointing those at risk of withdrawal, further refinements are still needed to improve sensitivity. By incorporating additional features or adjusting model parameters beyond what was done in the fourth model, it could be possible to create a more balanced and effective predictive model, ultimately enhancing its utility in recruitment decisions. A couple of recommendations can be made to improve this model:

1. **Model Improvement**: Experiment with more sophisticated models (e.g., ensemble methods) to potentially increase sensitivity without sacrificing specificity.
2. **Feature Engineering**: Consider including additional applicant characteristics or external factors (e.g., socio-economic status, academic performance trends) that might improve predictive accuracy.
3. **Recruiter Training**: Use the model's insights to prioritize high-risk applicants and optimize the recruitment efforts based on predicted outcomes.

IV. Decision Tree

For this project, we used a Decision Tree model to predict the completedadm variable, which indicates whether an applicant successfully completed all the steps of the admissions process. The process of creating this decision tree involved several stages, including data preparation, model training, evaluation, and enhancements to improve performance.

To begin, we prepared the dataset by selecting relevant features that we believed would influence an applicant's likelihood to complete the admissions process. These features included fgpa (a categorical version of GPA), schoolsel (school selectivity), major1group and major2group (groupings of undergraduate majors), appdeadline (application deadline month), attendedevent (whether the applicant attended an event), essayUW (categories of the number of unique words in essays), essentiment (essay sentiment), submitted (time of application submission), and completedadm (the target variable). These variables provided a comprehensive view of both academic and application-related aspects of each candidate.

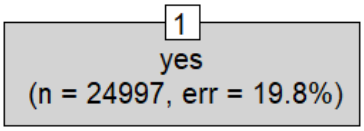
To evaluate our model's performance, we split the dataset into training and testing sets, using an 80/20 split. This split was carefully carried out to ensure that the original class distribution of completedadm was maintained in both the training and testing sets. Maintaining representative class distributions is crucial for ensuring that the model generalizes well and does not suffer from biases related to class imbalance.

```
> # Check class distribution in both sets
> proportions(table(train_DT$completedadm))

       no       yes
0.1983038 0.8016962
> proportions(table(test_DT$completedadm))

       no       yes
0.1982717 0.8017283
```

The initial Decision Tree model was trained using the C5.0 algorithm, which is known for its effectiveness in handling classification problems while providing an interpretable model. The initial model was trained using nine predictors and 24,997 samples. However, the resulting decision tree had a size of only one, indicating that the model was overly simplistic. This simplicity led to a high training error, suggesting that the model was unable to capture the complexities of the data and therefore struggled to make accurate predictions. The evaluation on the test dataset showed an accuracy of 80.17%. However, the model classified all observations as completedadm = 1, highlighting its inability to correctly identify instances of completedadm = 0. This issue was primarily due to the significant class imbalance, with approximately 80% of the samples being completedadm = 1.



Decision Tree Model 1: Confusion Matrix and Performance Statistics

Reference / prediction	0	1
0	0	1239
1	0	5010

- **Accuracy**: 0.8017 (95% CI : 0.7916, 0.8116)
- **No Information Rate**: 1
- **Kappa**: 0
- **McNemar's Test P-Value**: < 2e-16
- **Sensitivity**: NA (ability to identify applicants likely to withdraw)
- **Specificity**: 0.8017 (ability to identify applicants likely to complete)
- **Positive Predictive Value**: NA
- **Negative Predictive Value**: NA
- **Balanced Accuracy**: NA

Figure 4-1: Decision Tree Model 1 Confusion Matrix and Performance Statistics

To address the model's shortcomings, we attempted boosting, which is a method used to improve performance by iteratively training models and correcting the errors made by previous models. We attempted boosting with 100 trials, but due to the inaccuracies of individual classifiers, boosting was abandoned, and the model ended up producing identical predictions to the initial unboosted version.

In an effort to better handle the class imbalance, we introduced a cost matrix. This cost matrix placed a higher penalty on incorrectly predicting completedadm = 0 as completedadm = 1. Specifically, the cost of predicting "no" as "yes" was set higher than the cost of predicting "yes" as "no." Using this cost-sensitive approach, we retrained the decision tree model, which resulted in a more complex tree with 20 nodes. The most influential features in this updated model were essayUW, fgpa, and submitted.

```
> error_cost_DT
     no yes
no    1   2
yes   4   1
```

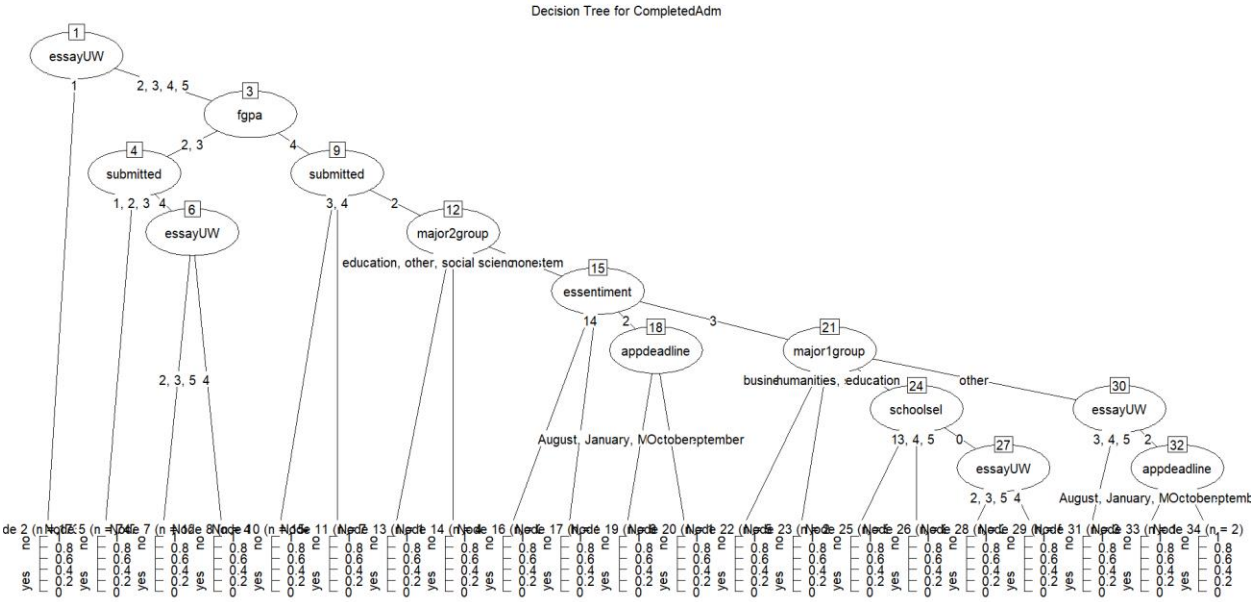


Figure 4-2: Decision Tree Model 1

Decision Tree Model 2: Confusion Matrix and Performance Statistics

Reference / prediction	0	1
0	24	1215
1	42	4968

- **Accuracy**: 0.7988 (95% CI: 0.7887, 0.8087)
- **No Information Rate**: 0.9894
- **Kappa**: 0.0171
- **McNemar's Test P-Value**: < 2e-16
- **Sensitivity**: 0.363636 (ability to identify applicants likely to withdraw)
- **Specificity**: 0.803493 (ability to identify applicants likely to complete)
- **Positive Predictive Value**: 0.019370
- **Negative Predictive Value**: 0.991617
- **Balanced Accuracy**: 0.583565

Figure 4-3: Decision Tree Model 2 Confusion Matrix and Performance Statistics

The cost-sensitive model showed slight improvements in classifying instances of completedadm = 0. On the test set, the model achieved an accuracy of 79.88%, and sensitivity improved to 36.36%, reflecting a better, albeit still limited, ability to identify the minority class. The balanced accuracy also showed improvement, reaching 58.36%, which indicates a better trade-off between sensitivity and specificity compared to the initial models.

The decision tree visualization clearly shows the key features and decision paths used in making predictions, illustrating how factors like the number of unique words in essays, GPA category, and submission timing influenced the final decision on whether an applicant completed the admissions process.

This iterative approach—starting from a simple model, evaluating its performance, attempting boosting, and finally applying a cost-sensitive method—allowed us to gradually refine our model to better handle the complexities of the dataset and address the challenges posed by class imbalance.

V. ANN

Report on ANN Models for Persona – Ideal Education Candidate

We removed the following variables before creating any models:

- ***personid***: Removed as it is a unique identifier and does not contribute predictive value to the model.
- ***appyear***: Removed since all entries belong to the same year (2015–2016), making it non-informative for distinguishing outcomes.
- ***schoolsel_chr***: Removed because it is redundant; the numeric variable *schoolsel* already represents the same information in a more usable format.

Variables for ANN Models – Ideal Education Candidate

- ***gpa_3.81***: Reflects academic preparedness and commitment.
- ***schoolsel_5***: Indicates motivation from a highly selective institution.
- ***major1_education_(general)***: Aligns with the persona's education focus.
- ***minor_psychology***: Adds relevance for education/social sciences roles.
- ***appdeadline_January***: Tracks deadline adherence and procrastination.
- ***attendedevent_1***: Reflects interest and engagement with the program.
- ***undergrad_uni_other_us_university***: Captures background diversity.
- ***submitteddate_120***: Measures late submission impact on commitment.

Reason for Omitting Other Variables

- Variables like **essay lengths, sentiment, and secondary majors** were excluded as they may not directly reflect commitment to the process. Additionally, all variables related to essays were removed as they are less relevant to understanding why candidates drop out of the application process.
- Overly general or non-informative variables (e.g., *signupdate* and *starteddate*) were replaced with the more meaningful *submitteddate*. Highly imbalanced or irrelevant categories (e.g., *minorgroup* = none) were removed to simplify the model and avoid

a. Model 1: ANN with Selected Predictors and No Hidden Layer

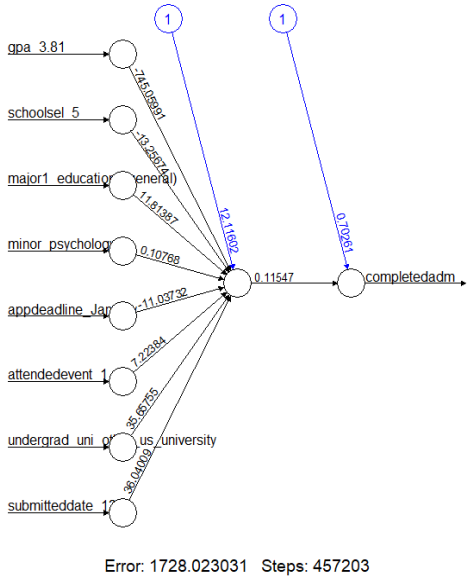
ANN Model 1: Confusion Matrix and Performance Statistics

Reference / prediction	0	1
0	1862	7511
1	0	0

- **Accuracy**: 0.1987 (95% CI: 0.1906, 0.2069)
- **No Information Rate**: 0.8013
- **Kappa**: 0
- **McNemar's Test P-Value**: < 2e-16
- **Sensitivity**: 1.0000 (ability to identify applicants likely to withdraw)
- **Specificity**: 0.0000 (ability to identify applicants likely to complete)
- **Positive Predictive Value**: 0.1987
- **Negative Predictive Value**: NaN
- **Balanced Accuracy**: 0.5000

Figure 5-1: ANN Model 1 Confusion Matrix and Performance Statistics

This model demonstrated high sensitivity, suggesting it was effective in identifying applicants who withdrew from the program. However, specificity was extremely low, failing to identify applicants likely to complete. The overall accuracy remained below the No Information Rate, indicating no improvement over random classification.



Model 2: ANN with Selected Predictors and 3 Hidden Nodes

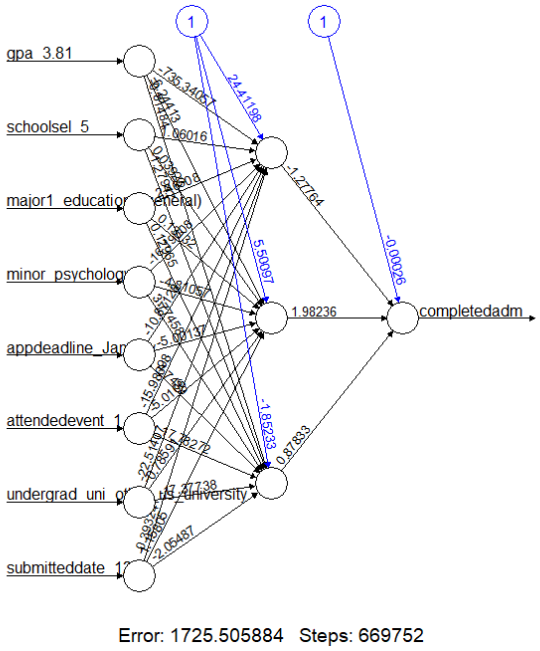
ANN Model 2: Confusion Matrix and Performance Statistics

Reference / prediction	0	1
0	1862	7511
1	0	0

- **Accuracy**: 0.1987 (95% CI: 0.1906, 0.2069)
- **No Information Rate**: 0.8013
- **Kappa**: 0
- **McNemar's Test P-Value**: < 2e-16
- **Sensitivity**: 1.0000 (ability to identify applicants likely to withdraw)
- **Specificity**: 0.0000 (ability to identify applicants likely to complete)
- **Positive Predictive Value**: 0.1987
- **Negative Predictive Value**: NaN
- **Balanced Accuracy**: 0.5000

Figure 5-2: ANN Model 2 Confusion Matrix and Performance Statistics

Adding three hidden nodes did not yield performance improvements. Sensitivity remained high, but specificity was still low, resulting in similar accuracy and balanced accuracy to the model without hidden layers.



Model 3: ANN with Selected Predictors and 2 Hidden Layers (5 Nodes Each)

ANN Model 3: Confusion Matrix and Performance Statistics

Reference / prediction	0	1
0	1862	7511
1	0	0

- **Accuracy**: 0.1987 (95% CI: 0.1906, 0.2069)
- **No Information Rate**: 0.8013
- **Kappa**: 0
- **McNemar's Test P-Value**: < 2e-16
- **Sensitivity**: 1.0000 (ability to identify applicants likely to withdraw)
- **Specificity**: 0.0000 (ability to identify applicants likely to complete)
- **Positive Predictive Value**: 0.1987
- **Negative Predictive Value**: NaN
  Balanced Accuracy: 0.5000

Figure 5-3: ANN Model 3 Confusion Matrix and Performance Statistics

Despite incorporating two hidden layers with five nodes each, this model did not improve upon previous configurations. Sensitivity remained perfect at identifying applicants likely to withdraw, but specificity was nonexistent, leading to a failure to predict completed cases. The overall accuracy and balanced accuracy remained consistent with prior models, falling well below the No Information Rate.

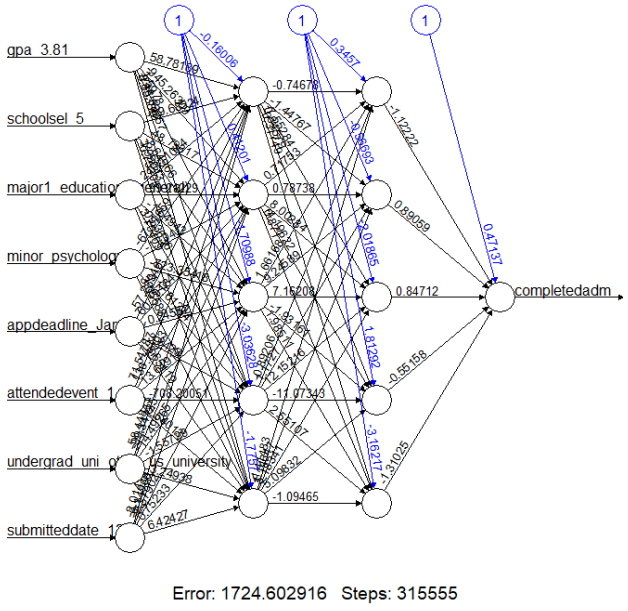


Figure 5-4: ANN

Conclusion and Recommendations

- **The model predicts all candidates as "withdrawn," resulting in:**
  True Negatives (1862): Correctly identified withdrawals.
  False Negatives (7511): Misclassified completions as withdrawals.
  Zero Specificity (0.0): Fails to identify completions.

This indicates:
Perfect Sensitivity (1.0): The model predicts everyone as "withdrawn," leading to a 100% sensitivity (it captures all class 0 cases).
Zero Specificity (0.0): It completely fails to identify completes (class 1).
Low Predictive Utility: The model predicts only one class (class 0), resulting in poor overall performance and no ability to distinguish between completed and withdrawn candidates.

- **Model Performance Imbalance**: All models (both with and without hidden nodes) displayed extremely high sensitivity but failed in specificity, indicating a consistent inability to identify applicants likely to complete. This suggests that the models may be overfitting to the " withdrawn " class, highlighting the need for balanced class representation or enhanced model complexity.
- **Limited Predictive Improvement**: None of the models outperformed the No Information Rate (NIR) of 0.8013, and the accuracy consistently fell below this benchmark. This indicates that the current feature set and model configurations are not effectively distinguishing between completed and withdrawn applicants. Experimenting with alternative features or introducing regularization may help improve predictive accuracy.
- This profile reflects a highly dedicated candidate who is strongly aligned with the program's objectives and has demonstrated significant engagement, though they may be more inclined to **withdraw** during the application process.

Recommendation:

- Explore additional or alternative features that may better capture “completed” risk.
- Evaluate other modeling approaches, such as decision trees or naive bayes methods, which might handle the complexity of this dataset more robustly than a basic artificial neural network.

VI. Support Vector Machines

a. Model 1: Mixed Features with Categorical and Numeric Predictors

Model 1 includes a mix of applicant academic information, essay characteristics, application deadlines, and event attendance status. The full set of variables used in this model are: *gpa, stem, schoolsel, major1group, major2group, minorgroup, essayuniquewords, essaysentiment,*

appdeadline, submitteddate, attendedevent, completedadm. These variables were factored appropriately to ensure compatibility with the SVM analysis.

Model 1: Confusion Matrix and Statistics

Reference / prediction	0	1
0	0	0
1	1239	5010

- **Accuracy**: 0.8017 (95% CI: 0.7916, 0.8116)
- **No Information Rate**: 0.8017
- **Kappa**: 0
- **McNemar's Test P-Value**: < 2e-16
- **Sensitivity**: 0.00000
- **Specificity**: 1.00000
- **Positive Predictive Value**: NaN
- **Negative Predictive Value**: 0.8017
- **Balanced Accuracy**: 0.5000

Figure 6-1: SVM Model 1 Results using Linear and RBF Kernel

The results for both the linear and RBF kernel models are shown above. While the accuracy is 80.17%, the model failed to predict any withdrawn applications, classifying all cases as completed. This led to a sensitivity of 0.000 and a specificity of 1.000. The kappa value of 0 and an undefined positive predictive value (NaN) further highlight the model's lack of predictive power. Despite the high accuracy, it merely reflects the imbalance in the dataset with a balanced accuracy of only 0.500 which indicates poor performance.

b. Models 2 and 3: Numeric Predictors

Since SVM models typically rely on numeric predictors, Models 2 and 3 were developed using only numeric variables. For Model 2, additional features such as essay lengths and key application dates were included. The full set of variables used for this model includes: *gpa, stem, schoolsel, essay1length, essay2length, essay3length, essayuniquewords, essaysentiment, signupdate, starteddate, appdeadline, submitteddate, attendedevent, and completedadm*.

Model 3 is similar but excludes the essay length variables, as they may not contribute meaningfully to the analysis and could be irrelevant for predicting the outcome. The variables included for this model are: *gpa, stem, schoolsel, essayuniquewords, essaysentiment, signupdate, starteddate, appdeadline, submitteddate, attendedevent, and completedadm.*

Model 2

Reference / prediction	0	1
0	6	0
1	1233	5010

Model 3

Reference / prediction	0	1
0	5	0
1	1234	5010

• **Accuracy**: 0.8027 (95% CI: 0.7926, 0.8125) • **No Information Rate**: 0.8017 • **Kappa**: 0.0077 • **McNemar's Test P-Value**: < 2e-16 • **Sensitivity**: 0.00484 • **Specificity**: 1.00000 • **Positive Predictive Value**: 1.00000 • **Negative Predictive Value**: 0.80249 • **Balanced Accuracy**: 0.50242	• **Accuracy**: 0.8025 (95% CI: 0.7924, 0.8123) • **No Information Rate**: 0.8017 • **Kappa**: 0.0065 • **McNemar's Test P-Value**: < 2e-16 • **Sensitivity**: 0.00403 • **Specificity**: 1.00000 • **Positive Predictive Value**: 1.00000 • **Negative Predictive Value**: 0.80237 • **Balanced Accuracy**: 0.50201

Figure 6-2: SVM Model 2 and 3 Results using RBF Kernel

For both Models 2 and 3, the linear kernel results were identical to Model 1, showing no improvement in performance. However, the RBF kernel analysis showed better outcomes compared to Model 1, with an increase in accuracy and positive predictive value. Model 2 achieved an accuracy of 80.27%, while Model 3 had a similar accuracy of 80.25%. Both models had perfect specificity (1.000) and a positive predictive value of 1.000. Despite these improvements, the sensitivity was still very low, at 0.00484 for Model 2 and 0.00403 for Model 3, showing the models struggled to identify within applications. Although the accuracy slightly increased, the balanced accuracy for both models highlights the issue of class imbalance and difficulty in predicting completed applications.

c. Model 4: Refined Numeric Predictors

Given the improved performance of Models 2 and 3, Model 4 was designed to be more targeted by excluding variables that might not be relevant to the analysis. For Model 4, the same variables from Model 3 were used except for essay lengths, signup date, and start date. The final set of variables included in this model are: *gpa, stem, schoolsel, essayuniquewords, essaysentiment, appdeadline, submitteddate, attendedevent, and completedadm.*

Model 4: Confusion Matrix and Statistics

Reference / prediction	0	1
0	7	0
1	1232	5010

- **Accuracy**: 0.8028 (95% CI: 0.7928, 0.8126)
- **No Information Rate**: 0.8017
- **Kappa**: 0.4195
- **McNemar's Test P-Value**: < 2e-16
- **Sensitivity**: 0.00565
- **Specificity**: 1.00000
- **Positive Predictive Value**: 1.00000
- **Negative Predictive Value**: 0.80263
- **Balanced Accuracy**: 0.50282

Figure 6-3: SVM Model 4 Results using RBF Kernel

Model 4 provided the best results across all iterations. It achieved an accuracy of 80.28%, slightly outperforming previous models by excluding less relevant predictors. The kappa value improved to 0.4195, indicating a stronger agreement compared to earlier models. The model maintained perfect specificity (1.000) and a positive predictive value of 1.000, with no false positives. While the sensitivity remained low at 0.00565, suggesting continued difficulty in identifying completed applications, the balanced accuracy increased to 0.50282. Overall, Model 4's refined feature selection led to a better performance.

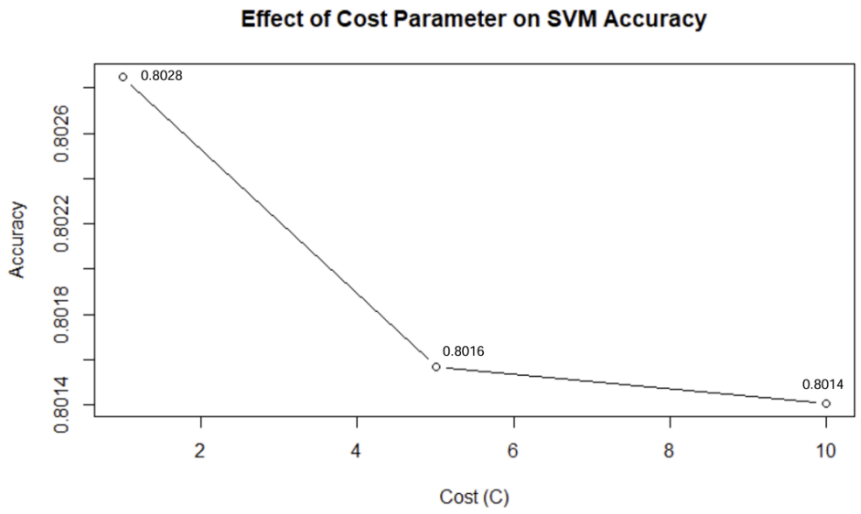


Figure 6-4: Cost Parameter vs Accuracy

Additionally, the cost parameter affects the model's accuracy, as seen in Figure 6-4. At lower values of C, the accuracy was highest at 80.28%. However, as C increased beyond 5, the accuracy started to drop, reaching 80.14% at C = 10. This decline suggests that higher cost values may have caused the model to overfit the training data, focusing too much on correct predictions rather than generalizing well. This shows that choosing an appropriate cost value is important, as setting it too high can hurt the model's overall performance.

d. Conclusion

Model 4 showed the best performance across all iterations. While Models 1, 2, and 3 struggled with low sensitivity and failed to correctly identify withdrawn applications, Model 4 improved by excluding less relevant features like essay lengths and certain date variables. This adjustment led to better overall accuracy and the highest kappa value, indicating a stronger model. Although the sensitivity issues persisted, Model 4 achieved a better balance between sensitivity and specificity. The analysis also underscored the need for careful tuning of the cost parameter, as higher values resulted in a noticeable decline in accuracy due to overfitting. Ultimately, Model 4 handled the class imbalance more effectively and provided the most reliable predictions.

VII. Results and Conclusion

Model	Accuracy	Sensitivity	Specificity	Kappa	Balanced Accuracy	Key Insights
Naive Bayes (NB)						
Model 1	0.792	0.06538	0.97166	0.0536	0.51852	High specificity, low sensitivity
Model 2	0.8004	0.024213	0.992415	0.0258	0.508314	Slightly higher accuracy, very high specificity
Model 3	0.8011	0.017756	0.99481	0.0197	0.506283	Similar accuracy, very high specificity
Model 4	0.7969	0.06457	0.97804	0.0625	0.52131	Cross-validation, consistent low sensitivity
Model 5	0.7968	0.06538	0.97764	0.0631	0.52151	Hyperparameter tuning, no substantial improvement
Decision Tree (DT)						
Model 1	0.8017	NA	0.8017	0	NA	Classified all as completed, high class imbalance
Model 2	0.7988	0.363636	0.803493	0.0171	0.583565	Cost-sensitive,

						improved sensitivity
Artificial Neural Network (ANN)						
Model 2-1	0.1987	1	0	0	0.5	High sensitivity, zero specificity
Model 2-2	0.1987	1	0	0	0.5	Adding hidden nodes did not improve performance
Model 2-3	0.1987	1	0	0	0.5	Two hidden layers, no improvement
Support Vector Machine (SVM)						
Model 1	0.8017	0	1	0	0.5	High specificity, zero sensitivity
Model 2	0.8027	0.00484	1	0.0077	0.50242	Improved accuracy, very low sensitivity
Model 3	0.8025	0.00403	1	0.0065	0.50201	Similar to Model 2, slightly lower sensitivity
Model 4	0.8028	0.00565	1	0.4195	0.50282	Best performance, refined feature selection

Discussion: Best Model

Based on the analysis of the four predictive models evaluated in this project—Naive Bayes, Decision Tree, ANN, and SVM—we compared their key performance metrics, including accuracy, sensitivity, specificity, and interpretability:

1. Naive Bayes: While the Naive Bayes models demonstrated high specificity across iterations, their sensitivity remained consistently low (below 0.07). This indicates a strong ability to identify applicants likely to complete but poor performance in recognizing those likely to withdraw the process. Despite its simplicity, this approach lacks the predictive power required for our goals.
2. Decision Tree: Decision Tree Model 2 emerged as the most balanced option. With a sensitivity of 36.36% and balanced accuracy of 58.36%, it provides a better trade-off between identifying complete and withdrawn applicants. The application of a cost-sensitive approach significantly improved its ability to classify the minority class. Moreover, its interpretability offers actionable insights for decision-making, making it suitable for operational use.
3. ANN: All ANN models failed to surpass the baseline No Information Rate (NIR). While they achieved perfect sensitivity (100%), their specificity was non-existent (0%), rendering them ineffective for this project. This suggests severe overfitting to the majority class and an inability to generalize.
4. SVM: Although SVM Model 4 achieved the highest overall accuracy (80.28%) and specificity (100%), its sensitivity was unacceptably low (0.00565). This makes it unsuitable for identifying applicants likely to withdraw the process, despite its high performance in other areas. Additionally, its lack of interpretability limits practical applicability for stakeholder use.

Given the project's objectives, Decision Tree Model 2 is the optimal choice due to its superior sensitivity, balanced accuracy, and interpretability, which align with the need to identify candidates likely to complete the admissions process effectively.

Conclusion

After a thorough comparison of all models, we concluded that Decision Tree Model 2 is the most appropriate for this project. Unlike the SVM Model 4, which excelled in accuracy but lacked sensitivity, Decision Tree Model 2 provides a more balanced approach with significantly higher sensitivity (36.36%) and interpretable decision paths. These strengths make it a valuable tool for predicting candidate behavior, particularly in identifying those likely to complete the process. Its cost-sensitive approach further addresses the class imbalance, improving its effectiveness in classifying the minority class.

Although there is room for improvement, such as exploring ensemble methods or additional features, Decision Tree Model 2 meets the immediate needs of the project. Its balance of predictive power and interpretability ensures actionable insights for refining recruitment strategies and supports Teach For America's goal of building a reliable pipeline of committed and diverse leaders. This model lays a strong foundation for further enhancements while delivering practical benefits in its current form.

VIII. Appendix

Naïve Bayes and Decision Tree Code:.

```
library(caret)
library(klaR)
library(C50)
library(kernlab)
library(dplyr)
library(fastDummies)
library(neuralnet)
library(ggplot2)
getwd()
cs <- read.csv("C:/Users/Carolina/Desktop/CHICAGO/IIT/Rcode/Proyect
PA/CaseStudy.csv")
#cs <- read.csv("CaseStudy.csv")
str(cs)
head(cs)
summary(cs)
View(cs)
##### DATA CLEANING-----------------------------------------------------------
----------------
csi <- cs[, -c(1, 2, 6)]
View(csi)

str(csi)
colSums(is.na(csi))
length(csi$schoolsel)
#we get rid of the NA in schoolsel

csi$schoolsel[is.na(csi$schoolsel)] <- 0
colSums(is.na(csi))
length(csi$schoolsel)
str(csi)
#transform to factor categorical variables
csi$stem <- factor(csi$stem)
str(csi$stem)
csi$schoolsel <- factor(csi$schoolsel) #being the level 0 the values that had
NA
str(csi$schoolsel)
csi$major1<- factor(csi$major1)
str(csi$major1)
csi$major2<- factor(csi$major2)
str(csi$major2)
csi$major1group<- factor(csi$major1group)
str(csi$major1group)
csi$major2group<- factor(csi$major2group)
str(csi$major2group)
csi$minor<- factor(csi$minor)
str(csi$minor)
csi$minorgroup<- factor(csi$minorgroup)
str(csi$minorgroup)

csi$undergrad_uni<- factor(csi$undergrad_uni)
str(csi$undergrad_uni)
csi$appdeadline<- factor(csi$appdeadline)
```

```
str(csi$appdeadline)
csi$attendedevent<- factor(csi$attendedevent)
str(csi$attendedevent)
csi$completedadm<- factor(csi$completedadm)
str(csi$completedadm)

csi$essay1[csi$essay1length >= 0 & csi$essay1length <= 75] <- 1
csi$essay1[csi$essay1length > 75 & csi$essay1length <= 150] <- 2
csi$essay1[csi$essay1length > 150 & csi$essay1length <= 225] <- 3
csi$essay1[csi$essay1length > 225] <- 4

# Check the result to confirm the categorization
table(csi$essay1)
csi$essay1<- factor(csi$essay1)
str(csi$essay1)

csi$essay2[csi$essay2length >= 0 & csi$essay2length <= 75] <- 1
csi$essay2[csi$essay2length > 75 & csi$essay2length <= 150] <- 2
csi$essay2[csi$essay2length > 150 & csi$essay2length <= 225] <- 3
csi$essay2[csi$essay2length > 225] <- 4

# Check the result to confirm the categorization
table(csi$essay2)
csi$essay2<- factor(csi$essay2)
str(csi$essay2)
csi$essay3[csi$essay3length >= 0 & csi$essay3length <= 75] <- 1
csi$essay3[csi$essay3length > 75 & csi$essay3length <= 150] <- 2
csi$essay3[csi$essay3length > 150 & csi$essay3length <= 225] <- 3
csi$essay3[csi$essay3length > 225] <- 4

# Check the result to confirm the categorization
table(csi$essay3)
csi$essay3<- factor(csi$essay3)
str(csi$essay3)

csi$essayUW[csi$essayuniquewords >= 0 & csi$essayuniquewords <= 100] <- 1
csi$essayUW[csi$essayuniquewords > 100 & csi$essayuniquewords <= 200] <- 2
csi$essayUW[csi$essayuniquewords > 200 & csi$essayuniquewords <= 300] <- 3
csi$essayUW[csi$essayuniquewords > 300 & csi$essayuniquewords <= 400] <- 4
csi$essayUW[csi$essayuniquewords > 400] <- 5

# Check the result to confirm the categorization
table(csi$essayUW)
csi$essayUW<- factor(csi$essayUW)
str(csi$essayUW)

# Redondear la columna essay_sentiment a un decimal
#csi$essayssentiment<- round(csi$essayssentiment, 1)
#head(csi$essayssentiment)
#csi$essayssentiment<- factor(csi$essayssentiment)
#str(csi$essayssentiment)

csi$essentiment[csi$essayssentiment <= -1] <- 1
csi$essentiment[csi$essayssentiment > -1 & csi$essayssentiment <= 0 ] <- 2
csi$essentiment[csi$essayssentiment > 0 & csi$essayssentiment <= 1] <- 3
csi$essentiment[csi$essayssentiment > 1] <- 4
table(csi$essentiment )
```

```
csi$essentiment<- factor(csi$essentiment)
str(csi$essentiment)

csi$signup[csi$signupdate  <= 100] <- 1
csi$signup[csi$signupdate > 100 & csi$signupdate <= 200] <- 2
csi$signup[csi$signupdate > 200 & csi$signupdate <= 300] <- 3
csi$signup[csi$signupdate > 300 & csi$signupdate <= 400] <- 4
csi$signup[csi$signupdate > 400] <- 5

# Check the result to confirm the categorization
table(csi$signup)
csi$signup<- factor(csi$signup)
str(csi$signup)

csi$started[csi$starteddate  <= 100] <- 1
csi$started[csi$starteddate > 100 & csi$starteddate <= 200] <- 2
csi$started[csi$starteddate > 200] <- 3

# Check the result to confirm the categorization
table(csi$started)
csi$started<- factor(csi$started)
str(csi$started)

csi$submitted[csi$submitteddate <= 50] <- 1
csi$submitted[csi$submitteddate > 50 & csi$submitteddate <= 100 ] <- 2
csi$submitted[csi$submitteddate > 100 & csi$submitteddate <= 150] <- 3
csi$submitted[csi$submitteddate > 150] <- 4

# Check the result to confirm the categorization
table(csi$submitted)
csi$submitted<- factor(csi$submitted)
str(csi$submitted)

csi$fgpa[csi$gpa <= 1] <- 1
csi$fgpa[csi$gpa > 1 & csi$gpa <= 2 ] <- 2
csi$fgpa[csi$gpa > 2 & csi$gpa <= 3] <- 3
csi$fgpa[csi$gpa > 3] <- 4

# Check the result to confirm the categorization
table(csi$fgpa)
csi$fgpa<- factor(csi$fgpa)
str(csi$fgpa)
str(csi)
colSums(is.na(csi))

#eliminate the cols that we dont want// just tranformed into categorical
csi <- csi[, !(names(csi) %in% c("gpa", "essay1length",
"essay2length","essay3length", "essayuniquewords", "essayssentiment",
"signupdate", "starteddate", "submitteddate" ))]
str(csi)
View(csi)

##### NAIVES BAYES----------------------------------------------------------
---------------

## create training and holdout sets
set.seed(1947)  # for reproducibility
```

```
# Split
idx <- createDataPartition(csi$completedadm, p=0.8, list=FALSE)  # From caret
pkg, read documentation
train.df <- csi[idx, ]
holdout.df <- csi[-idx, ]

# Check label distribution
proportions(table(csi$completedadm))
proportions(table(train.df$completedadm))
proportions(table(holdout.df$completedadm))

### run naive bayes using klaR package

# Use only origin and destination as predictors
fd.nb_od <- NaiveBayes(completedadm ~ ., data = train.df)

# Output model
fd.nb_od

# predict probabilities
fd.pred_od <- predict(fd.nb_od, newdata=holdout.df)

# Compute Confusion Matrices aka CrossTabs
confusionMatrix(fd.pred_od$class, factor(holdout.df$completedadm))

### run naive bayes using klaR package

# Use only origin and destination as predictors
fd.nb_od <- NaiveBayes(completedadm ~ stem + schoolsel + major1group +
major2group + minorgroup + appdeadline + attendedevent + essay1 + essay2 +
essay3 + essayUW + essentiment + signup + started + submitted + fgpa, data =
train.df)

# Output model
fd.nb_od

# predict probabilities
fd.pred_od <- predict(fd.nb_od, newdata=holdout.df)

# Compute Confusion Matrices aka CrossTabs
confusionMatrix(fd.pred_od$class, factor(holdout.df$completedadm))


### run naive bayes using klaR package
# Use only origin and destination as predictors
fd.nb_od <- NaiveBayes(completedadm ~ stem + schoolsel + major1group +
major2group + minorgroup + appdeadline + attendedevent + essay1 + essay2 +
essay3 + essayUW + essentiment  + submitted + fgpa, data = train.df)

# Output model
fd.nb_od

# predict probabilities
fd.pred_od <- predict(fd.nb_od, newdata=holdout.df)

# Compute Confusion Matrices aka CrossTabs
```

```
confusionMatrix(fd.pred_od$class, factor(holdout.df$completedadm))

# Setup traincontrol

# Cambiado para usar el dataset 'csi' y predecir 'completedadm'
tc = trainControl(p=0.8, method='cv', verboseIter = TRUE, number=10)

# Train model
model = train(csi[, colnames(csi)[colnames(csi) != 'completedadm']],
              csi$completedadm,
              method='nb', trControl=tc)
model
confusionMatrix(predict(model, newdata=holdout.df),
factor(holdout.df$completedadm))

## More extensive tuning

# Get tuning parameters
modelLookup('nb')

# Setup search grid
nbtune_grid <- expand.grid(usekernel = c(TRUE, FALSE),
                           fL = c(0, 0.5, 1),
                           adjust = c(0.5, 1, 1.5))
# Train model with tuning grid
model = train(csi[, colnames(csi)[colnames(csi) != 'completedadm']],
              csi$completedadm,
              method='nb', trControl=tc,
              tuneGrid = nbtune_grid)
model
confusionMatrix(predict(model, newdata=holdout.df),
factor(holdout.df$completedadm))

##### DECISION TREE-----------------------------------------------------------
----------------
csi2 <- csi[, c("fgpa", "schoolsel", "major1group", "major2group",
"appdeadline", "attendedevent", "completedadm", "essayUW", "essentiment",
"submitted")]
View(csi2)

# class distribution for 'Car_Cancellation'
table(csi2$completedadm)
which(colnames(csi2) == "completedadm")  # Check the index position

# Create a random sample for training and test data (80% training, 20%
testing)
set.seed(1947)  # For reproducibility
idx_DT <- createDataPartition(csi2$completedadm, p = 0.8, list = FALSE)

# Split the data into training and testing sets
train_DT <- csi2[idx_DT, ]
test_DT <- csi2[-idx_DT, ]

# Check class distribution in both sets
proportions(table(train_DT$completedadm))
proportions(table(test_DT$completedadm))
```

```
#TO CHECK WHERE IS THE COL COM CAR CANCELLATION
target_var_position <- which(colnames(csi2) == "completedadm")
target_var_position

## **Training a Basic Decision Tree Model**

# Train a simple decision tree model using C5.0
csi_model_DT <- C5.0(train_DT[-target_var_position], train_DT$completedadm)
csi_model_DT
summary(csi_model_DT)

# Plot the Decision Tree
plot(csi_model_DT, type = "simple", main = "Decision Tree for completedadm")
## **Evaluating Model Performance**

# Generate a confusion matrix to evaluate performance
csi_pred_DT <- predict(csi_model_DT, test_DT)
confusionMatrix(test_DT$completedadm, csi_pred_DT)

## **Improving Model Performance with Boosting**
# Train a boosted decision tree with 10 trials
# Boost the accuracy of decision trees
csi_boost10_DT <- C5.0(train_DT[-target_var_position], train_DT$completedadm,
trials = 100)
summary(csi_boost10_DT)


csi_boost_pred10_DT <- predict(csi_boost10_DT, test_DT)
conf_matrix_boosted_DT <- confusionMatrix(test_DT$completedadm,
csi_boost_pred10_DT)
print(conf_matrix_boosted_DT) #IT STAYS THE SAME WITH THE SAME PREDICTIONS
AND CONFUSION MATRIX

## **Applying a Cost Matrix**

# Define a cost matrix (adjust the values based on the importance of false
negatives/positives)

# Penalize errors for predicting 'no cancellation' when it should be
'cancellation'
error_cost_DT <- matrix(c(1, 4, 2, 1), nrow = 2,
                        dimnames = list(c("no", "yes"), c("no", "yes")))
error_cost_DT

# Ensure that the target variable is properly defined as a factor
train_DT$completedadm <- factor(train_DT$completedadm, levels = c(0, 1),
labels = c("no", "yes"))
test_DT$completedadm <- factor(test_DT$completedadm, levels = c(0, 1), labels
= c("no", "yes"))

# Train a decision tree model with the cost matrix
csi_model_cost <- C5.0(train_DT[-target_var_position],train_DT$completedadm,
costs = error_cost_DT)
summary(csi_model_cost)

# Evaluate the cost-sensitive model on the test set
csi_pred_cost <- predict(csi_model_cost,test_DT)
```

```
# Display the confusion matrix for the cost-sensitive model
confusionMatrix(test_DT$completedadm, csi_pred_cost)
plot(csi_model_cost, main = "Decision Tree for CompletedAdm")
```

ANN MODEL:

```
library(caret)
library(klaR)
library(C50)
library(kernlab)
library(caret)
library(dplyr)
library(fastDummies)
library(caret)
library(neuralnet)
getwd()
setwd("C:/Users/Smush/Desktop/P DJ Class")
OG<-read.csv("CaseStudy.csv")
str(OG)
summary(OG)

#ANN
ANNOG<- OG[,-c(1,2,6)]
ANNOG$schoolsel[is.na(ANNOG$schoolsel)] <- 0
#DUMMY DF (# convert to dummy variables (remove first dummy) for predictors)
ANNOG_D <- ANNOG %>%
  dummy_cols(select_columns = c( "gpa","submitteddate","schoolsel","major1",
"minor","appdeadline","attendedevent",
                                 "essayssentiment ", "undergrad_uni" ),
  remove_selected_columns = TRUE, remove_first_dummy = TRUE)

#TRAIN
set.seed(1947)
annidx <- createDataPartition(ANNOG_D $completedadm , p=0.7, list=FALSE)
anntrain <- ANNOG_D[annidx, ]
anntest <- ANNOG_D[-annidx, ]
names(anntrain) <- gsub(" ", "_", names(anntrain))
names(anntest) <- gsub(" ", "_", names(anntest))
names(anntest)
str(anntrain)
#-------------------------------------------------------------------------
-------------
#Persona  -   Education persona
#ANN 1&1
accOG_persona1 <- neuralnet(completedadm ~ gpa_3.81   + schoolsel_5
+major1_education_(general)+minor_psychology
                        +appdeadline_January
+attendedevent_1+undergrad_uni_other_us_university + submitteddate_120,
                        data = anntrain,stepmax = 1e6, linear.output = TRUE)
#Plot
plot(accOG_persona1)
pred.test_persona1<- predict(accOG_persona11, anntest)
class.test_persona1 <- apply(pred.test_persona1, 1, which.max)-1
# Confusion matrix
confusionMatrix(factor(class.test_persona1), factor(anntest$completedadm))
```

```
#ANN 3
#Model
accOG_persona3 <- neuralnet(completedadm ~ gpa_3.81   + schoolsel_5
+major1_education_(general)+minor_psychology
                            +appdeadline_January
+attendedevent_1+undergrad_uni_other_us_university + submitteddate_120,
                            data = anntrain,  hidden=3,stepmax = 1e6 )
#Plot
plot(accOG_persona3)
pred.test_persona3 <- predict(accOG_persona3, anntest)
class.test_persona3 <- apply(pred.test_persona3, 1, which.max)-1
# Confusion matrix
confusionMatrix(factor(class.test_persona3), factor(anntest$completedadm))

#ANN 5
#Model
accOG_persona5 <- neuralnet(completedadm ~ gpa_3.81   + schoolsel_5
+major1_education_(general)+minor_psychology
                            +appdeadline_January
+attendedevent_1+undergrad_uni_other_us_university + submitteddate_120,
                            data = anntrain,  hidden=c(5,5),stepmax = 1e6 )
#Plot
plot(accOG_persona5)
pred.test_persona5 <- predict(accOG_persona5, anntest)
class.test_persona5 <- apply(pred.test_persona5, 1, which.max)-1
# Confusion matrix
confusionMatrix(factor(class.test_persona5), factor(anntest$completedadm))
```